222-TP-008-001

ECS Client Database Evaluation

Technical paper - Not intended for formal review or Government approval.

June 1995

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

Lynne Case /s/ 6/28/95

Lynne Case, CIDM Prototype Lead Date
EOSDIS Core System Project

SUBMITTED BY

Craig Schiihahn /s/ 6/28/95

Craig Schillhahn, Release B Manager Date
EOSDIS Core System Project

Hughes Information Technology Corporation
Landover, Maryland

This page intentionally left blank.

Abstract

The ECS client tools have the need for various data access or organization functions. An example is the support for valid keywords and the dependencies between keywords (dependent valids). Some of the data dictionary service information will be cached locally upon starting the Earth Science Search Tool (ESST). This local cache will be queried to determine the appropriate valid keywords as the user selects and deselects options in the ESST. This is just one of the possible uses of a DBMS local to the client software. Since the client software will be installed on many user workstations, the client DBMS should be free to the user. An evaluation of public domain and shareware databases has been performed and this paper documents the results of this work.

Keywords: prototype, client, database, DBMS

This page intentionally left blank.

Contents

Abstract

1. Introduction

1.1 Purpose 1

1.2 Organization 1

2. Client Database Requirements

2.1 General Requirements 3

2.2 Requirement Definitions 3

2.2.1 Mandatory Requirements 3

2.2.2 Desired Requirements 4

3. Evaluation Process and Results

3.1 Evaluation Process 5

3.2 Evaluation Results 5

4. Summary

4.1 Recommended Implementation 8

Table

3-1. Client Database Results Summary 6

This page intentionally left blank.

1. Introduction

1.1 Purpose

The ECS client tools have the need for various data access or organization functions. An example is the support for valid keywords and the dependencies between keywords (dependent valids). Some of the data dictionary service information will be cached locally upon starting the Earth Science Search Tool (ESST). This local cache will be queried to determine the appropriate valid keywords as the user selects and deselects options in the ESST. This is just one of the possible uses of a DBMS local to the client software. Since the client software will be installed on many user workstations, the client DBMS should be free to the user. An evaluation of public domain and shareware databases has been performed.

This paper describes the requirements of the client DBMS and the evaluation process and results. The evaluation was performed as part of Release B Client, Interoperability, and Data Management Subsystems prototyping under the Client Database Prototype. This prototype was requested and the results reviewed by the Client subsystem developers. The results documented here reflect the concurrence of the Client subsystem developers and the prototype developer.

NOTE: This paper does not discuss the mechanisms of caching data dictionary information at the client. It also does not discuss the rationale behind this decision.

1.2 Organization

This paper is organized as follows:

Section 1 Introduction provides the purpose of the paper. Section 2 Client Database Requirements introduces the requirements of the client DBMS. Section 3 Evaluation and Results discusses the public domain and shareware DBMSs that were evaluated and the results of the evaluation. Section 4 Summary provides a summary of the recommendations.

Questions regarding technical information contained within this Paper should be addressed to the following ECS and/or GSFC contacts:

- ECS Contacts
  - Lynne Case, CIDM Prototype Lead, (301) 925-0359, lcase@eos.hitc.com
  - Kevin Limperos, Client Subsystem Lead (301) 925-0582, klimpero@eos.hitc.com

Questions concerning distribution or control of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Dr.
Landover, MD 20785

This page intentionally left blank.

2. Client Database Requirements

2.1 General Requirements

In general the requirements for the client database management system (DBMS) are that it: (1) uses relational model, (2) has an API (application programmer's interface) that can be accessed from C++, and (3) is easy to use. All three of these general requirements are related. There are some public domain C++ persistence storage packages available, but it was determined that these systems would require too much code for the benefit that it allowed. The whole purpose of the client database is to easily implement the efficient access of valid keywords and dependent valids information as well as other functions such as storage and sorting of search results. If the DBMS requires too much coding and learning curve, it defeats this purpose.

Since the client software must be hosted on a variety of workstations, the DBMS must also be portable. This is a very important aspect of the software that was also considered.

2.2 Requirement Definitions

2.2.1 Mandatory Requirements

The following requirements were mandatory for the client DBMS. Section 3 shows that many of the public domain DBMSs could be eliminated based on one or more of these requirements.

1. Supports the relational model -- The client DBMS should be a relational database in a loose use of the term. In other words, the DBMS will not be tested for the Dr. Codd's 12 rules of relational databases, but the DBMS should have a basic concept of "tables", "columns", and "rows". In some cases these had different names but they were of the same concept.
2. Create tables at run-time (API) - The programmer should be able to create new tables from the API.
3. Insert, update, and delete from run-time - The programmer should be able to insert, update, and delete data from the database through the API.
4. Perform natural joins on tables - The programmer should be able to submit queries that access data from more than one table in a single query.
5. Support for DISTINCT results - The query language should provide the capability to retrieve non-duplicate results from a table.
6. Support for sorting of results - The query language should provide the capability to sort the rows being returned from the database.
7. Higher level query language - There should exist some query language that allows the programmer to retrieve, insert, update, and delete data from the database easily.
8. API interface to database - There should be a well documented application programmer's interface to the DBMS.
9. Ports to many Unix platforms - The DBMS should be easily ported to many Unix platforms since the client software will be required on many Unix platforms.

2.2.2 Desired Requirements

The following criteria are desirable requirements for the client DBMS.

1. Low resource utilization - The database engine should not require excessive memory or disk space requirements.
2. Efficient performance - The database engine should provide efficient query methods to support response times that will appear reasonable to the user.
3. Outer joins - It is desirable to have a query language that supports queries such that you can select all the rows of a primary table and additional information from a secondary table. If the information from the secondary table does not exist, the primary table data is still selected.
4. Easy to use - The DBMS should be easy to use and have a short learning curve.
5. Sorting on multiple attributes - The query language should provide the capability to sort on multiple columns selected in a single query. For example, the programmer should be able to sort first within the instrument name and then the satellite name. (This is not a high priority and should probably have been put in the desirable requirements, but no harm was done since all packages were eliminated by some other mandatory requirement.)
6. Concatenated primary keys - It is desired that the primary key of a table could be made up of more than one column in the table.

3. Evaluation Process and Results

3.1 Evaluation Process

The USENET news group comp.databases contains a post of a "Catalog of Free Database Systems". This posting includes an extensive listing of public domain and shareware databases with a wealth of information such as the available interfaces, the query language, the use restrictions, a description of the product, robustness, etc. The list includes relational, object-oriented, deductive, and text DBMSs.

The first step in the process was to peruse this list for viable candidates. Many candidates were eliminated by the properties contained in the list. For example, all non-relational DBMSs were eliminated. Once a list of potential candidates were assembled, the software was downloaded and compiled. The data dictionary service database structure used in Prototype Workshop 1 was created in each of the candidate DBMSs. The data from the technical baseline was converted into the format for each database. This provided a result set of over 4800 rows when the entire dependent valids information was selected (combination of distinct collections, instruments, satellites, and geophysical parameters).

A user interface was constructed that was written in C++. This user interface was used to select valids lists and once the user selected values, the software selected the dependent valids for the subsequent lists. Performance of the queries was measured as well as ease of development of the database access.

3.2 Evaluation Results

From the initial scrubbing of the list of relational DBMSs, only three potential candidates emerged for evaluation: QDDB (Quick and Dirty Database), University INGRES, and mSQL (mini SQL). Each of these was downloaded to a Sun SPARC20 workstation running Solaris 2.3 and the databases created. Only mSQL passed the evaluation stage of accessing the database from the user interface. Table 3-1 summarizes the results of the client DBMS comparison of products versus requirements. Following the table a brief discussion of each package is provided.

Table 3-1. Client Database Results Summary

	DiamondBase	PQL	Qddb	Typhoon	Univ. INGRES	MetalBase*	mSQL	Postgres***	REQUIEM	shql
MANDATORY CRITERIA										
Create tables at run-time (API)			no		see note 1		yes			
Insert, update and delete from tables at run-time			yes		see note 1		yes			
Perform natural joins on tables			no		yes		yes			
Support for DISTINCT results		no	no		yes		yes			
Support sorting of results		no	yes		yes		yes			
Higher level query language	NONE		TCL/TK	NONE	QUEL		SQL			
API interface to database		NONE	yes		E-QUEL		yes			NONE
Ports to many Unix platforms			yes		no		yes		****	
DESIRED CRITERIA										
Low resource utilization			yes		yes		yes			
Efficient performance			yes		yes		note 2			
Outer joins			no		no		no			
Easy to use			no		yes		yes			
Sorting on multiple attributes			no		yes		yes			
Concatenated primary keys			no		yes		no			
* donations are requested										
** For commercial use requires $225 per user										
*** object-relational and more than what we need										
**** embedded version only on Mac										
NOTES:										
1. Could not get INGRES program written with E-QUEL to connect to database. It is supposed to work however.										
2. mSQL performs very poorly with join operations of tables of small size (> 100 rows)										

DiamondBase

This is a database written and accessed from C++. It does not contain a higher level query language and thus was eliminated. A schema compiler generates C++ class definitions and then these definitions are used to store and retrieve objects.

PQL

PQL (plain query language) provides rudimentary SQL queries. There is no API and no distinct operator and order by statement in the query language support.

Qddb

Qddb (Quick and Dirty Database) is a non-SQL based relational DBMS. The concepts behind it are not intuitive. Although it does have an interface to TCL/TK, it has no well documented API. A simple program was constructed to load data in the database by using one of the available utilities as a starting point. Without the API being documented, it would be very difficult to construct other programs to manipulate the data.

Typhoon

This relational DBMS was inspired by Raima's db-VISTA, but it has no query language. All access to the database is done through a C subroutine library.

University INGRES

This is the research version of the DBMS that inspired the commercial Ingres product. It is not portable and is no longer supported at the University of California at Berkeley. The source was compiled and the database engine with the terminal monitor could be used to construct queries, but the embedded QUEL programs would not connect to the database. Several messages and README files that were found corroborated the fact that there is a problem with EQUEL programs under anything but the original port (VAX/VMS).

MetalBase

This is a shareware package that was eliminated from consideration based on the fact that it does not appear to be portable to Unix platforms. It was also rated as being very inefficient.

mSQL

Mini SQL is a shareware package that is being developed and maintained as a backend database to a network management product called Minerva. It is free to universities and researchers, but is shareware to commercial companies. It was by far the most robust and easy to use packages of those brought into the EDF (ECS Development Facility). As part of a commercial package, it is likely to be well-maintained and bug-free.

Postgres

This is an object-relational DBMS developed at the University of California at Berkeley. It is highly sophisticated and over kill for the client DBMS application. This might be a good alternative in the future for more sophisticated applications at the client. The review states questions the stability of the package, however.

REQUEIM

REQUEIM is a relational DBMS based on a relational algebra query language called RQL. The embedded RQL version is only on the Macintosh and thus there is no API access on Unix. The review also questioned whether this package was being maintained.

4. Summary

4.1 Recommended Implementation

The recommended implementation is to use mSQL. This package is robust and seems to be relatively bug free. The performance of joins is questionable in terms of efficiency and high use of temporary disk space. More testing will be done to see if this is an issue of joining on non-key attributes or it is inherent in all join operations. The licensing issues must be worked out between the authors of the package and the ECS procurement office. It is hoped that a one-time fee can be paid in exchange for free distribution of binaries. If this arrangement cannot be worked out, Postgres is a viable alternative to be evaluated.